Riitta Välimaa

TUOTTEEN ALKUPERÄMAA

Java Servlet-tekniikka

Tekniikka
2017

VAASAN AMMATTIKORKEAKOULU
Tietotekniikka

TIIVISTELMÄ

Tekijä	Riitta Välimaa
Opinnäytetyön nimi	Tuotteen alkuperämaa
Vuosi	2017
Kieli	suomi
Sivumäärä	44 + 0 liitettä
Ohjaaja	Pirjo Prosi

Tämän työn tarkoitus on kehittää apuväline tuotteen alkuperämaan selville saamiseksi myyntimaassa, Danfoss Suomi-organisaatiossa. Sovelluksen tarkoitus on kartoittaa taajuusmuuttajan rakenteellinen prosenttiosuus EU:n sisämarkkina-alueella versus ulkomarkkina-alueella. Taajuusmuuttajan verotus on riippuvainen EU:n sisämarkkinaosuuden suuruudesta yrityksen myydessä sitä kuluttajille EU-rajojen sisällä. Tämän hetkinen käytäntö on saada tiedot esille manuaalisella päivityksellä Excel-taulukkoon.

Sovellus on rakennettu Java Servlet-tekniikan ympärille. Servlet generoi HTML-sivut asiakkaan kyselystä, sekä JSP-sivu generoidaan Excel-taulukoon käyttäjän niin halutessa. Tietokanta, jota käytetään on MS SQL Server-relaatiotietokanta. Tietokannassa on määritelty taulut ja sarakkeet, joita käytetään iScalan ylläpidossa. ERP-tuotannonohjausjärjestelmä on riippuvainen iScalan tiedoista.

Tietokannat, joista tiedot haetaan, sijaitsevat kahdella eri palvelimella ja tietokantatauluja on useita. Yksi tauluista on rakennettu hieman eri tavoin ja sijaitsee eri palvelimella kuin muut tietokannan taulut. MS SQL-tietokannan sarakkeiden otsikot ovat kirjain-numero-yhdistelmiä, joista ei suoranaisesti pysty lukemaan sarakkeisiin määriteltyä tietoa ilman, että tietää mistä ja mitä hakee, mutta muutama taulu on nimetty sisällöllisesti oikeilla nimillä.

Työlle asetetut vaatimukset ovat kyetä lukemaan taajuusmuuttajan rakenne-BOMista (Bill of Material) euromääräinen osuus, joka tulee EU:n ulkopuolelta ja EU:n sisältä, sekä molemmille prosentuaalinen luku.
Tuotteen kustannukset tuli jakaa seuraaviin kategorioihin: materiaalin kokonaiskustannukset, materiaalikustannukset sekä EU- että NonEU-alueelta. Kategorian määrittelee komponentin alkuperämaa.

Avainsanat alkuperämaa, sisä-ja ulkomarkkinat ja Java Servlet

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Tietotekniikka

ABSTRACT

Author	Riitta Välimaa
Title	Country of Origin of the Product
Year	2015
Language	Finnish
Pages	44 + 0 Appendices
Name of Supervisor	Pirjo Prosi

The purpose of this thesis was to develop a tool product to determine the country of origin in the country of sale, for Danfoss, Finland-organization. The aim of the application is to find out the structural percentage of frequency converters in the EU internal market area versus outside the market area. The taxation of frequency converter is dependent on the size of EU's internal market share when the company selling it to consumers within the EU borders. The current practice is to get information out of a manual update in the Excel table.

The application was built around the Java Servlet technology. Servlet generates an HTML page of the client survey, as well as the JSP page is generated onto the Excel-table if the user so wishes. The database used is a MS SQL Server relational database. Tables and columns used in the iScala maintenance are defined in the database. The ERP production control system is dependent on the information on iScala.

The database, on which information is sought, is located on two different servers and the databases have several database tables. One of the database tables is built in a slightly different way and it is located on a different server than any other database table. The column headings of the MS SQL database are letter-number combinations, and it is not possible to read information in the columns directly without knowing where and what to search for. But there are also a few tables the contents of which are named with correct names.

The requirements of the thesis were to be able to read the share in euros which comes from outside the EU and inside the EU, out of the Bill of Materials of the frequency converter and the percentage for both. The product cost was divided into the following categories: materials total cost, materials cost, both from the EU and non EU area. The category is defined by the country of origin of the component.

Keywords Country of Origin, internal and external market and Java Servlet

SISÄLLYS

KUVIO- JA TAULUKKOLUETTELO

LYHENTEET

StockCode	Varastolajimerkki Danfoss Vaasa-yhtiössä.
HTML	Hypertext Markup Language -sivuille tarkoitettu merkintäkieli.
UML	Graafinen mallinnuskieli
JDBC	Java ohjelmointirajapinta relaatiotietokantoihin.
JSP	Javaan perustuva menetelmä luoda HTML-sivuja.
PDM	Tuotetiedonhallintajärjestelmä
SAP	Toiminnanohjausalusta
ERP	Toiminnanohjausjärjestelmä
MRP	Materiaalin hallintajärjestelmä
DBMS	Tietokannan hallintajärjestelmä
DBS	Tietokantajärjestelmä
SQL	Standardoitu kyselykieli
API	Ohjelmointirajapinta
DTD	Yleinen tapa kuvata sovelluksen sanastoa ja kielioppia
JVM	Java virtuaalikone
BOM	Materiaaliluettelo
TKHJ	Tietokannan hallintajärjestelmä

1 JOHDANTO

Opinnäytetyö on kehitetty Danfoss Suomi organisaatiolle. Sovelluksen tarkoitus on saada nopeasti ja ketterästi yrityksen tuotteista alkuperämaan prosentuaalinen arvo. Tätä arvoa verrataan EU:n ulkopuoliseen myyntialueeseen ja lasketaan laitteen prosenttiosuus EU- ja NonEU-alueesta. Täten saadaan myyntialueiden EU- ja NonEU-osuus selville. Ohjelmistoa tulee käyttämään esimerkiksi Danfoss globaali liiketoimintakontrolleri (Business Controller), globaali tilaus- ja toimituskäsittelyosasto ja logistiikkaosasto (Order & Delivery Process & Logistics) sekä globaali liiketoimintaosasto (Business Intelligence). Tämän hetkinen tilanne on, että ohjelmassa on ainoastaan taajuusmuuttajaan liittyvää dataa, mutta mahdollista on myös laajentaa koskettamaan muita Danfoss-tuotteita.

Sovellus käynnistyy StockCode eli varastolajimerkin syötöllä kyselyruutuun. Sovellus hakee kyseisen varastolajimerkin sisältämät materiaalitiedot, valmistuskustannukset EU:n osalta ja myös NonEU:n osalta sekä näiden prosenttiosuus ja euromääräinen kustannus. Ohjelmisto suorittaa laskutoimituksia ja antaa tuloksen suoraan HTML-sivuille. Tarvittaessa käyttäjä voi ladata tiedot Excel-tiedostoon JSP-sivun kautta. Tuloksessa on laskettu EU- ja NonEu-osuus kunkin myyntilajimerkkiä vastaavan laitteen materiaalien valmistusmaan osalta.

Web-sovellus kokonaisuutena muodostuu servletistä, HTML-sivuista, JSP-sivusta ja apuluokista class- tai jar-tiedostoina. Käyttöliittymänä tässä työssä toimii HTML-dokumentti. Web-sovellusta käytetään HTML-protokollan yli selaimessa ja web-sovellus toimii web-palvelimella.

2 YRITYS DANFOSS

Danfoss on tanskalainen perheyritys, joka toimii globaalisti ympäri maailmaa eri maanosissa. Päämaja on Nordborg:ssa Tanskassa. Yritys on aloittanut toimintansa vuonna 1933. Danfoss on pörssissä noteeraamaton yritys, jonka omistavat ”Bitten and Mads Clausen Foundation”-säätiö. Yritys valmistaa kylmätuotteita, lämpötuotteita, teollisuusautomatiikkaa, taajuusmuutajia ja korkeapaineisia vesijärjestelmiä. Danfoss työllistää 25 292 henkilöä ja palvelee yli 100 maassa ympäri maailmaa. Tehtaita on 63 kappaletta 19 maassa. Vuonna 2015 Danfoss osti Vacon Oyj:n. Tällöin uusi liiketoimintayksikkö sai nimekseen Danfoss Drives, joka kuuluu Danfoss-konserniin. /1/

2.1 Tuotetiedon hallinta yrityksessä

2.1.1 iScala

Tuotetietoja hallitaan organisaatiossa monella eri järjestelmällä. Tässä työssä iScalan tietokannat näyttelevät suurinta roolia, koska haettava tieto sijaitsee siellä. Kaikki tauluihin menevä tieto tulee iScalan kautta.

iScala on monen järjestelmän tietovirran varastojärjestelmä. Kaikki tieto, mitä toisista järjestelmistä tulee, tallennetaan tämän järjestelmän kautta tietokantoihin. iScala on tärkeä työkalu koko organisaation toiminnanohjauksessa.

2.1.2 PDM

PDM (Product Data Management) on tuotetiedonhallintajärjestelmä, jossa hallitaan tuotetietoja koko niiden elinkaaren ajan. PDM on tärkeä osa tuotannon järjestelmiä, koska tuoterakenne on lähtökohtana kaikkien tuotantojärjestelmien toimintana. PDM:ssä sijaitsee kaikki tuoterakenteet.

2.1.3 ERP ja MRP

ERP (Enterprice Resource Planning) on toiminnanohjausjärjestelmä. Tämä on osa yrityksen liiketoiminnallisia prosesseja ja tietoteknistä ympäristöä. Tämä yhdistää järjestelmien tietovirrat yhteiseen tietokantaan. SAP ME (SAP Manufacturing

Execution) toimii toiminnanohjausalustana organisaatiossa. Järjestelmän kautta tuotanto kykenee valmistamaan laitteet suunnitellun standardin mukaisesti ja vaaditun rakenteen mukaiseksi.

MRP (Material Requirements Planning) toimii materiaalin hallintajärjestelmänä. Tämä järjestelmä liittyy komponenttien ja raaka-aineiden tarvelaskentaan.

2.1.4 Summium

Summium on asiakasrajapinta, jonka kautta tilaukset siirtyvät muihin järjestelmiin. Käyttäjä tekee tilauksen tämän järjestelmän kautta. Summium on yhteydessä niin PDM:ään kuin iScalaan.

3 NYKYTILA JA TAVOITTEET

Sovellus tulee käyttöön usealle Danfoss-osastolle. Sovellukselle on käyttöä varsinkin silloin, kun on tarve määritellä tuotteen tullietuuskohtelu. Sovellus auttaa myös pienentämään virhetilanteita laskutoimituksien yhteydessä.

3.1 Nykytila

Nykytilassa tiedon saaminen on manuaalista Excel-tiedostoon tulostamista. Tietoa saadaan, mutta tieto on raaka-dataa. Tietoa on paljon ja sitä on vaikea hallita. Turhaa tietoa sisältyy dataan ja se vie resursseja.

3.2 Tavoite

Tavoite on saada tarvittava data hallitusti, joustavasti ja nopeasti saataville silloin kun sitä tarvitaan. Sovelluksen tavoitteena on luoda sovellus, joka hakee ”napin painalluksella” tarvittavan tiedon. Tavoiteeseen kuuluu myös sovelluksen helppo kehittäminen myöhemmässä vaiheessa. Sovellus helpottaa esim. datan käyttöä, datan hakua, ajan hallintaa, virheiden karsimista laskutoimituksissa ja turhan datan karsimista.

3.3 Alkuperämaa ja tullietuuskohtelu

Alkuperämaa eli yleinen alkuperä tarkoittaa maata, jossa tuotteen valmistus suurimmalta osin tapahtuu. Yleisiä alkuperäsääntöjä käytetään silloin, kun alkuperä ei liity tullietuuskohtelun myöntämiseen. Tuotteen alkuperämaa on pakollinen tieto tuontitulli-ilmoituksessa. Sääntöjen perusteella kokonaan jossain maassa tuotetut tai valmistetut tuoteet ovat kyseisen maan alkuperää. Tuotteet, jotka valmistetaan useassa maassa, ei siis vain yhdessä maassa, ovat sen maan alkuperää, jossa tuotteen viimeinen merkittävä valmistus tai käsittely on suoritettu. Tuote voi saada eri alkuperän riippuen minkä maan alkuperäsäännöt sovelletaan tuotteeseen. Yleensä alkuperä määräytyy tuontimaan sääntöjen perusteella. /2/

Tullietuuskohtelu riippuu alkuperämaasta. Tuotteesta kannettava yleinen tulli voi olla alhaisempi tai sitä ei peritä lainkaan, kun tuontituote on tietyn maan alkuperätuote. Alkuperä täytyy osoittaa alkuperäselvityksellä. Yrityksen on pyydettävä itse etuuskohtelua tulli-ilmoituksella. Alennettua tullikohtelua kutsutaan myös preferenssikohteluksi. /2/

Danfoss Drives Vaasa seuraa ns. 70/30-sääntöä, jossa 30 % käytetystä materiaalista voi tulla EU:n ulkopuolelta.

4 TIETOKANNAT

Tietokannat ovat toisiinsa liitettyjä tietoja järjestettyssä kokoelmassa. Tiedon on oltava sisällöltään moitteetonta, ajan tasalla, helposti saatavilla, oikean muotoista, ylläpito täytyy olla helppoa, suojattu asiattomilta ja turvassa vahingoittumiselta.

4.1 Tietokannan hallintajärjestelmä

Tietokantoja hallitaan DBMS, -ja Database Management System-hallintajärjestelmän avulla. Suomalainen lyhennys on TKHJ eli tietokannan hallintajärjestelmä. Hallintajärjestelmä on tarkoitettu suurten ja monen käyttäjän tietokantojen käsittelyyn. Tämä järjestelmä mahdollistaa tietokannan perustamisen ja ylläpidon.

Järjestelmä hoitaa:

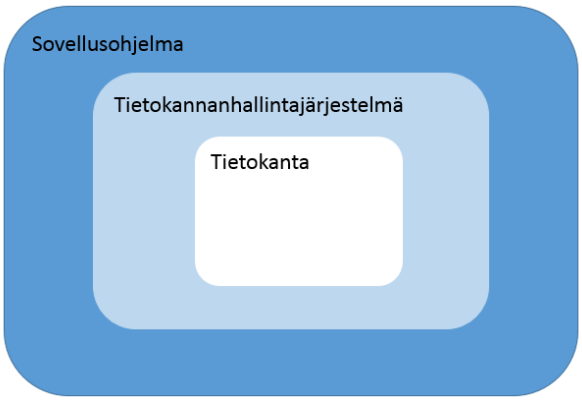
- tiedon jakamisen, tiedon viennin ja haun
- toiston jakamisen
- monen käyttäjän samanaikaisia toimenpiteitä
- transaktioita
- SQL eli kyselykieltä
- varmuuskopiointia ja tietojen palauttamista
- käyttöoikeuksien jakamista
- käyttöliittymien mahdollistamisen
- tietojen suhteiden esittämisen
- eheys-sääntöjen valvomisen.

Tietojärjestelmän arkkitehtuuri liittyy hallintajärjestelmään. Tämä arkkitehtuuri kuvaa rakenneosia, ulospäin näkyviä ominaisuuksia, sekä niiden välisiä yhteyksiä ja riippuvuuksia. Tämä muodostaa rungon järjestelmän suunnittelulle ja toteutukselle sekä mahdollisuuden järjestelmän kehittämiseen ja ylläpitoon sen elinkaaren ajan. Tietokannan riippumattomuus määritellään tietokantakaavion (schema) avulla. Tietokannan määrittely on ohjelmariippumaton ja myös usein ohjelmointikieliriippumaton. Taulujen samoja tietoja voidaan käyttää moneen

tarkoitukseen, mutta scheman avulla voidaan kohdistaa tieto oikeaan kohteeseen. /3/

4.2 Tietokantajärjestelmä

Tietokantajärjestelmä (Database System, DBS). Tietokantajärjestelmä muodostuu tietokannasta, tiedonhallintajärjestelmästä ja käytettävästä sovellusohjelmasta (Kuvio 1.).



Kuvio 1. Tietokantajärjestelmän kerrokset

Nämä yhdessä muodostavat kokonaisuuden, jolla voidaan luoda tietokantahakuja halutulla tavalla. /4/

4.3 Relaatiotietokanta ja sen rakenne

Relaatiotietokantamalli on tunnetuin rakennetason tietomalli. Tunnetuimmat relaatotietokannat ovat esim. IBM DB2, Oracle, Sybase, Informmix ja Microsoft SQL Server. Relaatiomalli koostuu kolmesta osasta, jotka ovat: rakenne, käsittely ja eheytyssäännöt. Relaatiotietokanta tarkoittaa tietokannan perustuvan tauluihin ja niiden välisiin suhteisiin. Tietokannan taulut perustuvat kenttiin ja tietueisiin, jossa kenttä koostuu sarakkeista ja tietue riveistä. Näiden avulla saadaan rakennettua tietoa tiettyihin kokonaisuuksiin sekä luodaan suhteita alkioiden välille. Alkioilla tarkoitetaan jakamatonta tietokokonaisuutta. /4/

Relaatiomallissa tietokanta koostuu tietoalkioiden muodostamista relaatioista. Taulussa on sarakkeita (attribuutti/kenttä) ja rivejä (monikko/tietue). Sarakkeille

määritetään tietotyyppi, joka on numeerinen tai merkkimuotoinen sekä tietotyyppille määritellään pituus. /4/

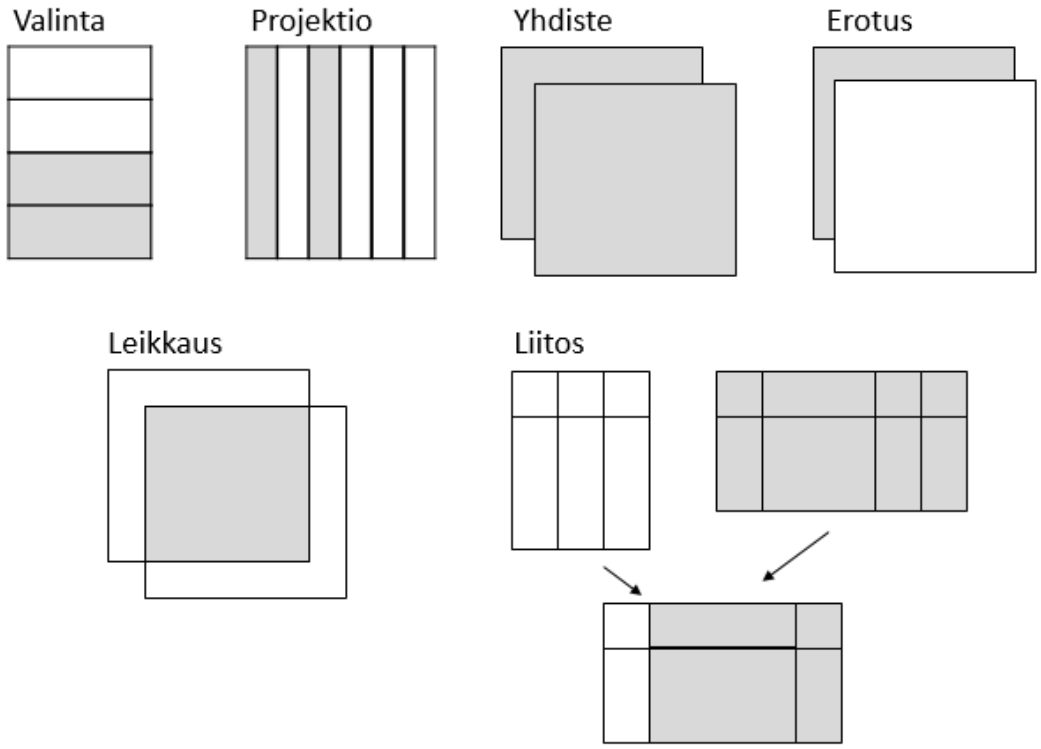
Perusavain (primary key, PK) on taulussa tunnisteena kaikissa riveissä. Tämän avulla identifioidaan taulun rivi ja sen tiedot riippuvaiseksi toisistaan. Muilla riveillä ei voi olla samanlaista arvoa perusavaimen sarakkeessa. Toisessa taulussa voi olla viiteavain, joka viittaa toisen taulun perusavaimeen. Viittaavaa taulua kutsutaan lapsitauluksi ja viittauksen kohteena olevaa taulua isätauluksi. Viiteavaimet ovat tarpeellisia yhdistettäessä taulujen tietoja toisiinsa eli tehdessä liitoksia. /4/

Viite-eheys tarkoittaa, että isätaulusta löytyy perusavain, johon lapsitaulussa viitataan. Isätaulusta ei voi poistaa riviä, jos lapsitaulun kyseinen rivi jäisi tyhjäksi. /4/

4.4 Joukko-oppi

Relaatiomalli perustuu IBM:n tutkija E.F.Coddin julkaisemaan relaatiomalliin (The relational model). Tämä malli määrittelee relaatiotietokantojen teoreettisen pohjan. Tämä teoreettinen pohja perustuu joukko-oppiin, matematiikkaan sekä predikaattilogiikkaan. /3/

Coddin teorian pohjalta tietoja käsitellään joukko-opillisesti. Taulu muodostuu joukosta rivejä ja näihin voidaan kohdistaa joukko-operaatioita. Joukko-operaatio voi kohdistua koko tauluun tai useampaankin tauluun. Tällä opilla voidaan tehdä myös päivityksiä. Joukko-oppi toteutetaan SQL-kielellä, joka perustuu joukko-oppiin. Kaikki operaatiot voidaan tehdä SQL-kielellä, SELECT-määreen eri muodoilla (Kuvio 2.). /3/



Kuvio 2. Joukko-operaatioita. /3/

4.5 Tietokannan suunnittelu lyhyesti

Tietokannan suunnittelussa täytyy huomioida, että se on hyvin ja perusteellisesti suunniteltu. Tässä on lueteltu tärkempiä vaiheita suunnittelussa:

- Vaatimusten kerääminen
  - Alustavat vaatimukset.
- Käsiteanalyysi
  - Tehdään karkeat piirrustukset tietokannasta, luokka- tai käsitekaavio.
- Tarveanalyysi
  - Täydennetään ja testataan tehtyä käsitemallia.
- Normalisointi
  - Tarkistetaan, että malli kolmannessa normaalimuodossa.
- Taulujen muodostaminen
  - Käsitemalli muunnetaan relaatiokannan taulurakenteiksi.
- Suorituskyvyn tarkistaminen

    - Lisätään indeksit ja muut mahdolliset toimenpiteet.
- Toteutus ja testaus.

On hyvä muistaa, että turha, toistuva tieto vie tilaa. Ylläpidon täytyy olla ketterä ja nopea. /3/

Tiedon kartoitukseen on hyvä tehdä esimerkiksi ER-kaavio. Kartoitetaan attribuutit ja niiden väliset riippuvuudet. ER-kaavio kuvaa järjestelmän tietoja, joiden tilojen täytyy pysyä muuttumattomina ohjelman suorituskertojen välillä. Seuraava askel on loogisten rakenteiden suunnittelu. Tässä sijoitetaan yhteenkuuluvat tiedot samaan tauluun, sitten askel eteenpäin teknisten rakenteiden suunnitteluun eli fyysinen suunnittelu. /3/

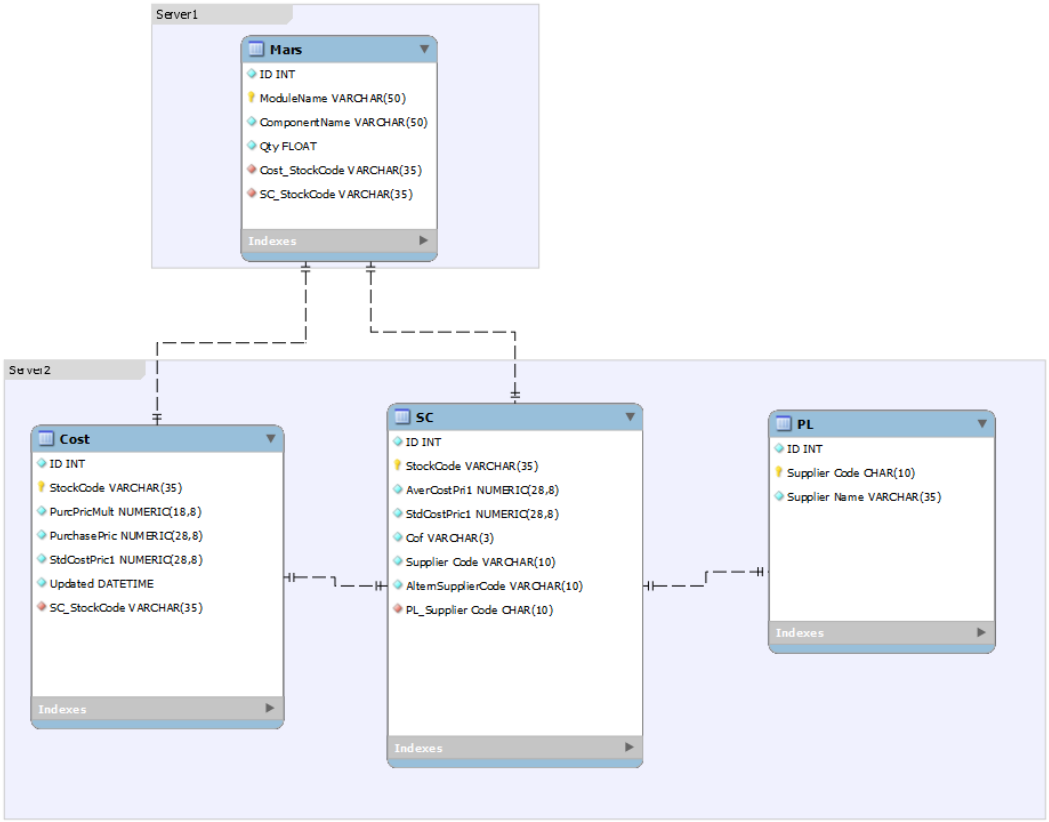
4.6 Sovelluksen relaatiotietokanta

Sovellus käyttää tietolähteenä relaatiotietokantaa MS SQL Server Databases (Microsoft), joka sijaitsee kahdella eri palvelimella. Tässä työssä käytetään valmiita tietokantoja ja tauluja.

Taulujen rakenteet ovat valmiita rakenteita. Taulut ja sarakkeet on nimetty osalta merkkijonomaisesti, esim. CV010100, mutta myös kuvaavalla nimellä nimettyjä tauluja löytyy, esim. StockCode. Sovellus käyttää tietolähteenä kahta relaatiotietokantaa, jotka molemmat ovat MS SQL Server Databases (Microsoft)-tietokantoja. Yksi tietokannoista sijaitsee eri palvelimella kuin muut tietokannat. Yhteys on luotu JDBC-ohjelmointirajapinnan avulla.

4.7 ER-kaavio

ER-kaaviota käytetään tietokantojen suunnittelussa ja mallintamisessa. Tämän työn tietokannat ovat jo valmiita tietokantoja, mutta ER-luokkakaavio kuvaa taulut ja taulujen suhteita toisiinsa. /3/



Kuvio 3. ER-kaavio tietokannan tauluista.

Kaaviossa on havaittavissa suhteet toiseen tietokantatauluun. Taajuusmuuttajia on monia variaatioita, mutta jokaisella taajuusmuuttajan variaatiolla on uniikki juokseva numero lajimerkissä. Taajuusmuuttaja koostuu monesta komponentista. Komponentilla on hinta, joka riippuu alihankkijan hinnasta ja ohjaa myös laitekustannuksia ja -myyntihintaa. Alihankkijoita voi olla yhdelle materiaalille yksi tai useampi, mutta yhdellä alihankkijan materiaalilla voi olla vain yksi hinta.

Käsitekaavio käsittää samanlaisten ominaisuuksien omaavia asioiden joukkoa. Käsitteiden väliset yhteydet kertovat siinä tarvittavat viiteavaimet. Ideana on hahmottaa aihealueen keskeiset käsitteet ja niiden väliset suhteet. Tällä kuvataan tietojen ja käsitteiden rakennetta, missä käsitteet kuvataan laatikoissa ja nuolella osoitetaan niiden väliset suhteet. (Kuvio 3.) /3/

Alkuperämaa on se maa, jossa kyseinen komponentti on valmistettu suurimmalta osin. Maa on haettu tietokannasta kolmen kirjaimen lyhenteenä, esim. FIN. Tämä

merkkijono määritellään binääriluvuksi 1 tai 0, riippuen onko maa EU:n sisä- vai ulkomarkkinamaa. EU-maat ovat binääriluku 0 ja EU:n ulkopuoliset maat ovat binääriluku 1. Tämän toimituksen avulla saadaan euromääräinen osuus molemmista kohteista. Myös prosentuaalinen osuus molemmista kohteista saadaan samalla tavalla. /2/

Alkuperämaata voi olla niin monta kuin on maitakin, mutta yhdellä tietyllä komponentilla voi olla vain yksi alkuperämaa sitä valmistamassa maassa. /2/

4.8 SQL-kieli

SQL, Structure Query Language, on standardoitunut relaatiotietokantatoimittajien tietokantakieleksi. SQL-kielen perusidea on määritellä mitä tehdään, ei miten tehdään. Tästä syystä SQL:ää voidaan käyttää useimmissa tietokannoissa. /4/

SQL-lauseilla luodaan tauluja ja muutetaan taulun sisältöä. Tauluista voidaan poistaa, lisätä, muuttaa ja palauttaa tietoa SQL-kielen avulla. Sen avulla määritellään käyttäjät ja heidän oikeudet tietokantaan. Myös yksittäisiin tauluihin saadaan käyttäjät ja oikeudet. /4/

Tietokannan erilaiset käskyt tehdään seuraavasti SQL-kielen:

- CREATE luodaan taulu
- ALTER muutetaan taulua
- DROP poistetaan taulu
- SELECT kysellään tietoja taulusta tai tauluista
- INSERT lisätään tietoa tauluun
- UPDATE päivitetään tietoa riville tai sarakkeeseen
- DELETE poistetaan tietoa, esim. rivi tai arvo
- GRANT luodaan oikeudet, esim. tiettyn tauluun
- REVOKE evätään oikeudet, esim. tiettyyn tauluun
- COMMIT tapahtuma, joko suoritetaan kokonaan tai ei ollenkaan
- ROLLBACK perutaan koko tapahtuma.

Rivirakenne on vapaa eli siinä voidaan käyttää isoja ja pieniä kirjaimia. SQL-kieleen varatut sanat voidaan kirjoittaa sekä isoilla että pienillä kirjaimilla. Jos lauseita on useita peräkkäin, ne täytyy erottaa puolipisteellä toisistaan, ja lause yleisesti loppuu puolipisteeseen. /3/

SELECT-lause koostuu tietyistä osista, jotka täytyy olla lauseessa mukana sekä tietyssä järjestyksessä toimiakseen. Lausejärjestys on seuraavanlainen:

- SELECT
- FROM
- WHERE
- GROUP BY
- HAVING
- ORDER BY.

SELECT-määre on esimmäisenä, koska tämä määre hakee halutut sarakkeet. FROM on seuraava määre, joka selvittää mistä taulusta haetaan kyseiset sarakkeet. WHERE-määre osoittaa mitkä rivit haetaan kyseisistä sarakkeista. GROUP BY-määre ryhmittelee rivit siten, että yhdessä ryhmässä on kaikilla riveillä sama arvo. Tämä määre ottaa osaa myös, jos sarakkeessa on yksi tai useampi koostefunktio (MIN(),MAX(),COUNT(),SUM(),AVG()). Jos SELECT-määreessä on yksi tai useampi koostefunktio, GROUP BY-määreeseen lisätään silloin sellaiset sarakkeet, jotka eivät ole koostefunktioiden argumenttejä.

HAVING-määrettä voidaan käyttää tarkentaessa vielä GROUP BY-määreen tulosjoukkoa. HAVING-määreellä voidaan karsia tulosjoukon tuplariviä sekä voidaan asettaa ehtoja ryhmille, kuten GROUP BY-määre asettaa ehtoja riveille. HAVING-määreessä on sallittua käyttää myös koostefunktioita. ORDER BY-määre määrittelee miten lajitellaan. Järjestys on se, missä järjestyksessä sarakkeet ovat määreessä. Määre voidaan lajitella nousevalla järjestyksellä (ASC), joka on myös oletuksena määreessä tai laskevalla järjestyksellä (DESC). Nämä lyhenteet sijoitetaan lajiteltavan sarakkeen perään. /4/

SQL:ssä on viisi valmista koostefunktiota. Funktioilla voidaan laskea sumeerisista sarakkeista summia SUM(), minimiä MIN(), maksimia MAX() ja keskiarvoja AVG(). Koostefunktiot eivät hae yksittäisiä rivejä, vaan laskevat joukosta rivejä tunnuslukuja. /4/

Jos tiedot ovat jakautuneet useaan tauluun ns. normalisoitu, tai tietoja on yhdisteltävä monesta taulusta, tämä voidaan tehdä liitosten JOIN-avainsanan avulla tai WHERE-lauseen ehtoon perustuen, mitä kutsutaan perinteiseksi liitossyntaksiksi. Kuviossa 2 kuvataan näitä liitoksia, jotka perustuvat joukko-operaatioihin. /4/

Muita liitoksia ovat INNER JOIN-liitos. Tässä liitoksessa JOIN-osiossa on liitettävä taulu ja ON-osiossa on liitosehto. Tämä liitos ottaa mukaan vain ne rivit, joille löytyy vastine toisessa taulussa. OUTER JOIN-liitoksessa on samoin JOIN-osiossa liitettävä taulu ja ON-osiossa liitosehto. Tämä liitos eroaa INNER JOIN-liitoksesta kuitenkin siten, että liitos ottaa myös sellaiset rivit mukaan liitoksessa, mitkä eivät täytä JOIN-ehtoa. LEFT [OUTER] JOIN tarkoittaa, että vasemmalla on taulu, josta otetaan kaikki tieto mukaan tietojoukkoon. RIGHT [OUTER] JOIN tarkoittaa, että oikealla on taulu, josta otetaan kaikki tieto mukaan tietojoukkoon. /4/

Tauluja voidaan yhdistää myös UNION-määreen avulla. Yhdisteen avulla voidaan yhdistää myös useampia tauluja. Tässä laitetaan useita SELECT-lauseita, joihin asetetaan UNION-määre väliin. Tämä yhdiste käyttäytyy kuten DISTINCT-määre, eli estää saman rivin toistumisen tuloksessa. /4/

Tauluille voidaan antaa viitenimet FROM-osiossa, joita täytyy myös käyttää SELECT-osiossa. Esim. k.koira, jossa k on viitenimi ja koira on sarakkeen nimi. /4/

Alikyselyillä on mahdollista rajata pääkyselyä kirjoittamalla pääkyselyn sisään SELECT-käsky. Sisäkkäisiä alikyselyita voi olla monta peräkkäin. Alikyselyn suoritus alkaa alimmalta tasolta. Alikyselyissä voidaan käyttää koostefunktioita. Alikyselyitä on myös seuraavanlaisia IN- , ANY- ja ALL-alikysely. Jos

alikyselyssä käytetään vertailuoperaattoria, tällöin kysely voi palauttaa vain yhden arvon, kuten koostefunktio. IN-, ALL-, ANY-operaattorin ehdon kanssa voidaan palauttaa monta arvoa. Alikysely on suluissa ja sisältää vain yhden sarakkeen.

DISTINCT-määre on alikyselyssä oletuksena mukana. EXIST-operaattorilla saadaan selville ovatko tietyt rivit olemassa vai eivät. Voidaan käyttää myös NOT EXIST-operaattoria, se antaa tulokseksi joko tosi/true tai epätosi/false. Alikyselyn tulos täytyy olla ainakin yksi rivi, että tulos olisi tosi. /4/

Jos tulos on NULL, silloin myös tulokseksi kirjautuu NULL. Jos NULL-arvoa ei haluta tulokseen, käytetään COALESCE-funktiota. Tämä funktio korvaa NULL-arvon toisella arvolla, esim. COALESCE(arvo,0). Esimerkissä NULL korvataan numeerisella nollalla. /4/

Tässä esitetyt asiat ovat vain osa SQL-kielessä esiintyvistä metodeista. SQL-kieli on monipuolinen ja sen avulla voidaan tehdä paljon asioita, jotka voidaan silloin jättää ohjelmointikoodista pois. /4/

5 SOVELLUS

Työssä on käytetty Java Servlet-tekniikkaa. Tekniikka mahdollistaa mm. tietokantahakujen tulosten esittämisen HTML-sivuilla käyttäjän tarpeiden mukaisesti. Java Servlet ladataan kertaalleen Web-palvelimen muistiin, josta sitä käytetään toistuvien pyyntöjen suorittamiseen. /6/

Servlettien avulla luetaan palvelupyynnön mukana tulevaa tietoa, joka tulee esimerkiksi WWW-lomakkeilta sekä palvelupyyntöön sisällytetystä tiedosta.

5.1 Monikerrosarkkitehtuuri

Järjestelmät suunnitellaan usein monikerrosarkkitehtuurin mukaisiksi, missä sovellus hajautetaan useaan komponenttiin ja nämä voivat olla eri palvelimilla. Komponentit ovat asiakaskerros (client tier), liiketoimintakerros (business tier) sekä tietokantakerros (database tier). /6/

Kaksikerrosarkkitehtuurissa komponentit on hajautettu asiakas- ja palvelinkerrokseen. Asiakaskerros koostuu käyttöliittymästä ja muusta ohjelmakoodista, palvelinkerros koostuu tietokantapalvelimesta. Tässä mallissa asiakaskerros hoitaa sekä esityslogiikan että liiketoimintalogiikan, ainoastaan tietokantakerros on palvelimella. Kyseinen malli ei ole joustava muutosten osalta. Ongelmana on käyttöliittymän ylläpito ja sen jakelu on raskasta. /6/

Monikerrosarkkitehtuuria kutsutaan myös kolmikerrosarkkitehtuuriksi. Tässä arkkitehtuurissa kaikki kolme komponenttia ovat erikseen. Asiakaskerros sisältää esityslogiikan eli ainostaan käyttöliittymän. Liiketoimintalogiiikkakerros sisältää sovelluslogiikan. Tietokantakerros sisältää ainoastaan tietokannan. Tällöin asiakaskerros on kevyt käsitellä ja ylläpito helppoa. /6/

Asiakaskerros voidaan toteuttaa selaimessa HTML-sivuina. Tämä voidaan toteuttaa Java-Servletillä tai JSP-sivuilla. Sovelluslogiikka voidaan toteuttaan Java-servletillä, joka toimii Web-palvelimella. Tietokantakerros toteutetaan relaatiotietokantana. /6/

5.2 Työssä käytetty kolmikerrosarkkitehtuuri

Sovelluksessa on käytetty kolmikerrosarkkitehtuuria. View-pakkaus sisältää Java Servletin, Controller-pakkaus sisältää ModuleDao-luokan sekä Model-pakkaus Module-luokat. Työssä on myös käytetty JSP-sivua asiakkaan halutessa tiedot Excel-taulukkoon.

ModuleDao-luokassa suoritetaan kaikki tietokantatoimenpiteet, Module-luokkien ollessa olioiden ominaisuuksien varastoluokkina. Module-luokat luetaan liiketoimintalogiikkakerroksiksi. Dao-luokka luetaan kolmiarkkitehtuurissa tietokantakerrokseksi. ModuleDao-luokka ohjaa yhteyttä tietokantapalvelimelle sekä sisältää SQL-lauseen, joka lähetään suoritettavaksi tietokantaan. Servlet-luokka on esityslogiikkakerros, jossa näytetään asiakkaalle tulos visuaalisena HTML-sivuilla.

5.2.1 Sovelluksen luokkakaavio

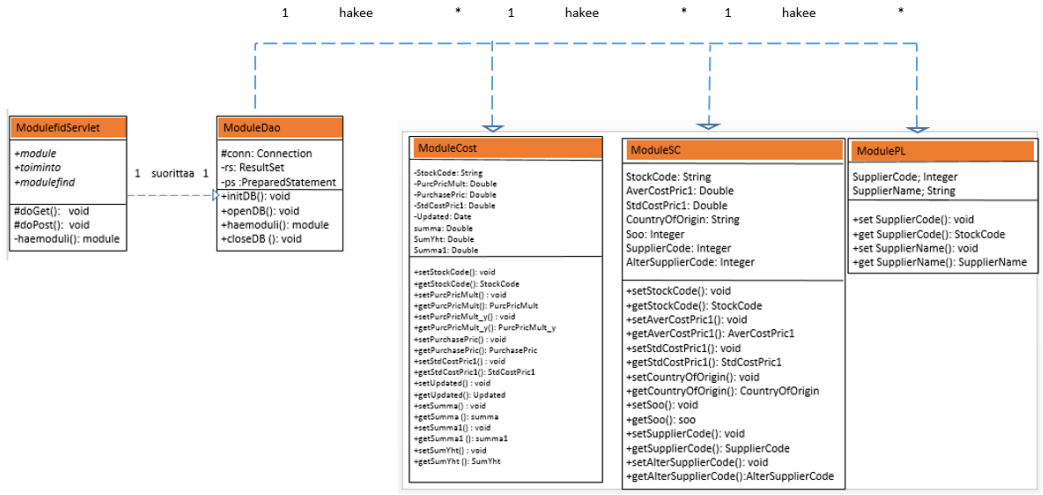
Luokkakaaviossa esitetään luokkia, attribuutteja, metodeja ja yhteyksiä. Luokkakaaviolla kuvaillaan sovelluksen luokkia ja niiden välisiä yhteyksiä toisiinsa. Yhteyksiin on liitetty myös lukumääräsuhteita, jotka kuvaavat kuinka monta viereisen luokan oliota voi liittyä toiseen luokkaan sekä mitä yhteys tekee. Mikäli yhteyden suunta halutaan kuvata, se voidaan kuvataan nuolenkärjellä. Lukumääräsuhteilla, kuten ”N..M”, tarkoitetaan kuinka monta oliota vähintään (N) ja enintään (M) voi liittyä tiettyyn olioon yhteyden toisessa päässä. Tätä kokonaisuutta kutsutaan assosiaatioksi. /5/

Puhuttaessa koostumussuhteista, niitä on kahdenlaisia. Vahvassa koostumus-suhteessa molempien luokkien olioiden elinikä on saman pituinen. Oliot luodaan ja tuhotaan samalla kertaa. Heikkossa koostumussuhteessa oliot säilyvät vaikka yhteys purkaantuu. /5/

Sovelluksessa on luokat kuvattu omissa laatikoissaan, missä näkyy luokan nimi, siihen liittyvät attribuutit ja metodit, jotka on eritelty omiin laatikoihin luokkanimen alle. Attribuuttien ja metodien näkyvyyden merkintä on toteutettu plus (+)- ja miinus (-)–merkkien sekä #-merkin avulla. Plus-merkki on public-

näkyvyys, miinus-merkki on private-näkyvyys sekä #-merkki on protected. Attribuuttien ja metodien perässä olevä määre tarkoittaa palautetyyppiä, esimerkiksi String, joka palauttaa merkkijonon.

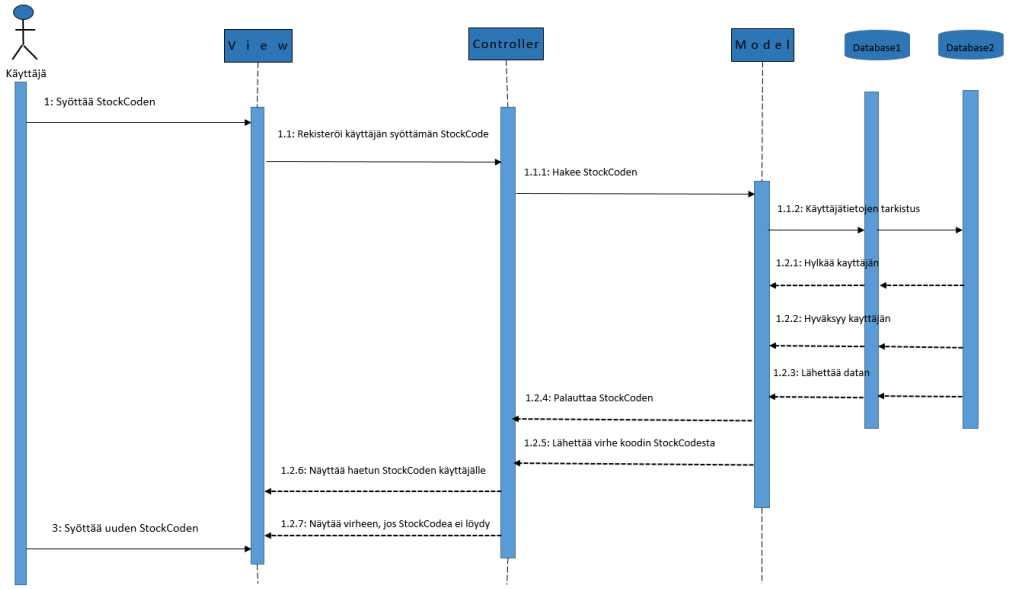
ModulefindServlet-luokka on servlettiluokka, joka on sovelluksen ydin luoden tulosjoukosta HTML-sivut. ModuleDao-luokka generoi yhteyden tietokantaan ja lähettää kyselylauseen sinne. ModuleCost, ModuleSC ja ModulePL ovat luokkia, jotka varastoivat oliot tulosta varten. ModuleDao-luokkaan liitty monta eri oliota, joten luokkien välinen suhde on 1..*. (Kuvio 4.) /6/



Kuvio 4. Sovelluksen luokkakaavio.

5.2.2 Sovelluksen sekvenssikaavio

Sekvenssikaaviossa (tapahtumakaavio) kuvataan eri osapuolien vuorovaikutusta keskenään. Kuviossa 5 on kuvattu käyttäjän ja sovelluksen välinen kommunikointi. Kuviossa on otettu myös huomioon poikkeukset.



Kuvio 5. Sovelluksen sekvenssikaavio.

Käyttäjä syöttää StockCoden HTML-sivuille. Controller-osio hakee Model-osion välityksellä tietokannoista halutun datan. Käyttäjän tunnistus tapahtuu serverillä, tämä tarkoittaa sovelluksessa Windows-autentikointia, autentikointi joko hylätään tai hyväksytään. Hyväksytyllä autentikoinnilla saadaan data controllerin kautta servletille ja sieltä HTML-sivuille näkyviin. Käyttäjä voi hakea heti uuden StockCoden halutessaan. (Kuvio 5.).

5.3 Java Servlet

Java Servlet on laitteistoriippumaton laajennus web-palvelimelle, joka sisältää Java-ohjelman tai –ohjelmia. Java Servletit voivat toimia omassa kehyksessään esim. JavaServer Faces, Vaadin. Kehyksessä voidaan suorittaa erilaisia toimintoja, esimerkiksi tietokantakyselyjä. Palvelupyynnöt suoritetaan käyttäen säikeitä, jolloin prosesseista tulee kevyempiä. /4/

Java Servletit pystyvät keskustelemaan Web-palvelimen kanssa, jolloin tämä mahdollistaa servlettien välisen kommunikoinnin. Web-palvelimena käytetään tässä sovelluksessa Apache Tomcat-sovelluspalvelinta, koska ohjelmointi on toteutettu Java-kielellä. /4/

Tietoturva on tässä ohjelmassa toteutettu suojatulla resurseilla. Kun ohjelmaa ajetaan, tarkistetaan sekä myönnetään käyttäjän oikeudet ajaa ohjelmaa.

5.4 Servletin elinkaari

Kun servletiä kutsutaan ensimmäisen kerran, se latautuu palvelimen muistiin. Muistista se on saatavilla, kunnes se halutaan tuhota. Metodit init() ja destroy() ovat elinkaaren suhteen tärkeitä metodeja. Service()-metodi ohjaa pyynnöt.

Init()-metodia kutsutaan, kun servlet ensimmäisen kerran ladataan. Silloin servlet latautuu palvelimen muistiin. Destroy()-metodia käytetään, kun halutaan servletin poistuvan muistista lopullisesti. Näistä toiminnoista, latauksesta ja poistosta, huolehtii javax.servlet.servlet-rajapinta. Tämä rajapinta määrittää nämä metodit, joita voidaan kutsua eri aikoina tai tietyssä järjestyksessä. /4/

Servlettien elinkaari koostuu seuraavanlaisesti:

- lataaminen
- alustus
- palvelu
- tuhoaminen
- roskien keruu.

5.5 Servlet Container

Servlet Container huolehtii servletin lataamisesta sekä poistamisesta, mutta se huolehtii myös pyyntöjen välittämisestä asiakkaalta palvelimelle ja palvelimelta asiakkaalle. Container on tietynlainen ”moottori” servletille, mitä ilman ei servlet voisi toimia. Tässä työssä ”moottorina” toimii Apache Tomcat. /7/

5.6 Servlet API

API (Application programming interface) on rajapinta, joka mahdollistaa eri ohjelmien keskustelun keskenään.

Servlet Api on ohjelmointirajapinta. Se määrittelee, miten Java-luokka kirjoitetaan sekä mitä metodeja toteutetaan, jotta se voi toimia servlettinä. Tomcat-palvelin tukee Servlet Api:a, joten servlet voidaan kirjoittaa, kääntää ja suorittaa palvelinkoneella. Tämä rajapinta koostuu kahdesta eri pakkauksesta: javax.servlet- ja javax.servlet.http-pakkauksesta. javax.servlet määrittelee rajapinnat, jotka liittyvät servletteihin ja sen luokkiin. javax.http määrittelee rajapinnat, jotka liittyvät http-protokollan käyttäytymiseen ja sen luokkiin.

Servlet-rajapintoja ovat Generic Servlet, Servlet Request, Servlet Response ja Servlet Context.

Generic Servlet toimii kaikkien servlettien kantaluokkana. Server Context:n olion avulla servletti on vuorovaikutuksessa ympäristöönsä.

HttpServletRequest- ja HttpServletReaponse-olion avulla välitetään ja palautetaan palvelupyyntö. Kun servlet-palvelin vastaanottaa kyselyn, esim. http-lomakkeelta, luodaan siitä HttpServletRequest-olio. Tämä viedään servletille sitä kutsuttaessa. Palautukseen käytetään HttpSevletResponse-oliota, joka palautetaan Requestiin. Näiden luokkien avulla servlet käsittelee pyyntöjä ja vastauksia. Nämä ovat laajennettuja luokkia ServletRequest- ja ServletResponse-luokista.

Sovelluksessa näitä on hyödynnetty siten, että request:sta luetaan HTML-lomakkeen tiedot HttpServletRequest rajapintaa käyttäen. Haetaan tiedot tietokannasta ja luodaan HTML-sivut HttpServletResponsea responsea käyttäen.

Servlet API myös määrittelee miten attribuutteja asetetaan servletin requestiin, http-sessioon ja web-sovelluksen kontekstiin. /7/

5.7 doGet()- ja doPost()-metodit

HTTPServlet-luokka määrittelee doGet()- ja doPost()-metodit. void doGet (HttpServletRequest request, HttpServletResponse response) on doGet()-metodin esittelymuoto. void doPost(HttpServletRequest request, HttpServletResponse response) on doPost()-metodin esittelymuoto. Parametri request sisältää tietoa http-pyynnöstä, kun taas parametri response sisältää tietoa http-vastauksesta.

Selain voi lähettää doGet-metodin avulla web-palvelimelle rajoitetun määrän tietoa URL:n yhteydessä. Tämä on HTTP-protokollan oletusmenetelmä. doPost()-metodilla selain voi lähettää web-palvelimelle rajoittamattomasti tietoa yhdellä kertaa. Tätä metodia käytetään HTML-lomakkeen tietojen lähettämisessä palvelimelle. /7/

5.8 Request

Request on asiakkaan palvelupyyntö, jonka HttpServletRequest-objekti kapseloi kaiken tiedon. Http-protokollan mukaan tieto on tallennettu joko http-otsikkotietoihin tai viestin runkoon. /7/

Requestiin voidaan liittää parametreja ja attribuutteja. Molemmat ovat nimi-arvo-pareja. Erona näissä on, että atribuutin arvo voi olla mikä tahansa Java-objekti eli ei rajoitu vain merkkijonoihin kuten parametri. /7/

Attribuutteja voidaan myös käyttää, kun requestin käsittely ohjataan toiselle servletille tai JSP-sivulle. Attribuutteja ei voida asettaa HTML-sivuilla, kuten parametreja, vaan niiden käsittely tapahtuu servletissa tai JSP-sivuilla. /7/

Requestissa käytetään attribuuttien käsittelyssä seuraavia metodeja: /7/

public object getAttribute(String name);

public Enumeration getAttributeNames();

public void setAttribute(String name,Object obj);

Sovelluksessa on käytetty setAttribute()-metodia siirrettäessä tietoa servletiltä JSP-sivulle request.setAttribute("module", module)-metodilla. JSP-sivulla tieto otetaan vastaan kokoelmaan, ArrayListiin. Tiedon läpikäynti JSP-sivulla tehdään for-each-silmukalla ja tulostetaan Excel-tiedostoon

```
RequestDispatcher rd = request.getRequestDispatcher("excel.jsp");
rd.forward(request, response);
```

RequestDispatcher-rajapintaa on käytetty hyväksi tiedon siirtämiseen JSP-sivulle. Dispatcher-objektin avulla voidaan ohjata servletin käsittely eteenpäin toiselle servletille tai JSP-sivulle.

request.setAttribute(”omistaja”, new omistaja());

Tässä metodissa attribuutti saa uuden arvoksi uuden omistaja-olion. Aikaisempi attributti pyyhkiytyy pois ja uusi arvo tulee sen tilalle. /7/

5.9 Response

Servletille tuleva palvelupyyntö voi sisältää vaikka kuinka monta parametria. HTML-lomakkeet sisältävä usein paljon tietoa. Parametrit ovat nimi-arvo-pareja, joissa merkkijono name on avain arvoon. Esimerkkinä sovelluksen lomakkeesta yksi hakulomake.

```
out.println("<form action='ModulefindServlet' method='get'>");
out.println("<td>Enter the StockCode: <input size=\"40\"type='text'
name='ModuleName'/></td><br>");
out.println("<input type='submit' name='toiminto' value='Search' />");
out.println("</form>");
```

Tässä ylläolevassa esimerkissä, ohjataan tekstikentän ”name” sisältö parametrina ModulefindServlet-servletille. HTML-sivuilla osoitteesta voi nähdä parametrin seuraavasti: http://localhost:8080/Coo/ModulefindServlet?ModuleName=xxxxxxx

ServletRequestiin liitettyjä parametrejä voidaan lukea metodeilla:

public String getParameter(String avainsana);

String nimi=Request.getParameter(“name”);

Tässä haetaan parametrit muuttujiin. Parametri-name-arvo haetaan nimi-muuttajaan. Työssä on haettu tällä tavoin parametrejä HTML-lomakkeesta. /7/

5.10 Servlet Context

java.servlet.ServletContext on rajapinta eli Web-sovelluksen konteksti määrittelee servleteille näkymän Web-sovellukseen. Tämän avulla servletit voivat kirjata

tapahtumia loki-tiedostoihin. Tämä mahdollistaa viittaukset resursseihin sekä kaikille saman kontekstin sisällä toimiville servleteille käyttää yhteisiä muuttujia. /7/

Konteksti on sidottu aina johonkin tiettyyn hakemistopolkuun palvelimella. Konteksti voi sijaita esimerkiksi osoitteessa ”http//localhost/soo”, tämä on kontekstipolku. Apache Tomcat-ympäristössä webapps-hakemisto sisältää kaikki kontekstit eli Web-sovellukset. Jokaisella on oma web.xml-tiedosto, jonka avulla voidaan määritellä palvelimelle konteksteja. /7/

5.11 Web.xml

Web.xml sijaitsee Apache Tomcatin weapps-hakemistossa sovelluksen kontekstihakemiston alla olevassa WEB-INF-kansiossa. Kontekstia hallitaan web.xml-tiedoston ja rajapinnan tarjoamien metodien avulla.

Web.xml-asetustiedoston avulla määritellään miten sovellus käyttäytyy ja minkälaisia resursseja, komponentteja ja alustusparametreja eri komponentit käyttävät.

Tiedoston avulla määritellään esimerkiksi seuraavia asioita:

- servlettien nimeäminen
- servlettien URL-polut
- servlettien alustusparametrit
- session asetukset
- staattisten tiedostojen MIME-tyypit
- tietoturva-asetukset.

Web-tiedoston täytyy olla validi, kuten kaikkien xml-tiedostojen. Tämä tarkoittaa, että jokaisella määrityksellä täytyy olla alku- ja loppuelementit. Tiedoston alussa pitää olla <?xml version=”1,0” encoding=”ISO-8859-1”?>-merkkaus. /4/

5.12 Annotaatio

Servlet Api 3-version annotaatioilla voidaan kirjoittaa sovelluksen määritykset ohjelmakoodiin mukaan. Näillä voidaan antaa tietoa mm. Java-kääntäjälle. Sovelluksessa on käytetty annotaatioita, esimerkiksi servletissä @WebServlet("/ModulefindServlet"). Annotaatioit alkavat @-merkillä ja niiden perässä on suluissa olevia elementtejä. Ne kirjoitetaan luokkien ja luokan jäsenten määrittelyjen eteen omalle riville. /4/

5.13 Java Server Pages JSP

Ohjelmassa on käytetty servletin tukena JSP-tekniikkaa (Java Server Pages). Halutessa tiedot voidaan tulosta Excel-taulukkoon. JSP-tekniikka hyödyntää Servlet rajapintaa, jossa on HTML-merkinnän seassa Java-koodia. Java-koodi erotetaan omilla merkeillä <%java koodia %>. /5/

ContentType määrittelee Excel-tiedoston tyypin sovelluksessa xls-tiedostoksi. JSP-sivulla on mahdollisuus määritellä tiedosto myös uudemmaksi xlsx-tiedostotyypiksi. /5/

```
response.setContentType("application/vnd.ms-excel");
response.setHeader("Content-Disposition", "attachment; file-
name=cheet.xls");
request.setAttribute("module", module)
```

Servlet suorittaa tietokantahaun haeModuli()-metodilla ModuleDao-luokasta, sekä generoi tuloksen HTML-sivuille.

Sivulla on mahdollisuus syöttää tiedot JSP-sivun kautta Excel-tiedostoon. Asiakkaan syöttäessä tietynlaisen StockCoden hakuruutuun, sovellus hakee tietokannasta määrätyt tiedot käyttöliittymän näytölle. Käyttäjä voi halutessaan siis ajaa tiedot Excel-taulukkoon. /5/

5.14 CSS-sivut

CSS-sivut sisältävät HTML-sivujen ulkoasun määrittelyn. Tämän tyylitiedoston avulla on määritelty servletistä generoidun HTML-sivujen ulkonäkö.

5.15 JDBC API- ja JDBC-ajuri

Java Database Connectivity eli JDBC on universaali ja toimittajariippumaton standardi, jonka mukaan Java-sovellukset voivat kommunikoida tietokantojen kanssa. Jokaisella tietokantatoimittajalla on omat toteutukset JDBC-rajapinnan sisäiseen toteutukseen omiin tietokantoihin, mikä johtaa omiin JDBC-ajureihin. /8/

JDBC (Java API) (Application Programming Interface) eli ohjelmointirajapinta tarjoaa kehyksen käsitellä relaatiopohjaista dataa, kuten relaatiotietokantoja. APIn runko sijaitsee java.sql-paketissa ja javax.sql-paketissa on uudemmat ominaisuudet. /8/

JDBC avulla voidaan suorittaa SQL-lausekkeita tietokantaan.

JDBC-rajapinta tukee seuraavia ominaisuuksia:

- yhdenaikaisia yhteyksia useampaan tietokantaan
- tapahtumien (transaktioiden) hallintaa
- yksinkertaisia kyselyjä
- tallennettuja proseduureja (ohjelmia, jotka tallentuvat tietokantaan)
- esikäännettyjen lauseiden hyödyntämistä
- sidottuja muuttujia
- pääsyä tietokannan määrittelytietoihin
- kursoria.

JDBC APIn tärkeimmät luokat löytyvät java.sql-luokasta. Näitä ovat esimerkiksi DriverManager, Connection, Statement sekä ResultSet.

- java.sql.DriverManager lataa ohjelman yhteyden tarvitsemat ajurit.
- java.sql.Connection generoi Java-oliorajapinnan tietokantaan.
- java.sql.Statement mahdollistaa SQL-lauseiden suorittamisen kyseisen yhteyden avulla.
- java.sql.ResultSet on Java-rajapinnan kautta saatu tulosjoukko.

Muita käytettyjä luokkia ovat myös SQLException, PreparedStatement. java.sql.SQLExeption on tärkeä halutessa ohjelman käyttävän Java poikkeusten käsittelyä. Poikkeukset voidaan tämän avulla käsitellä hallitusti ilman, että sovellus pääse kaatumaan virhetilanteen sattuessa. /5/

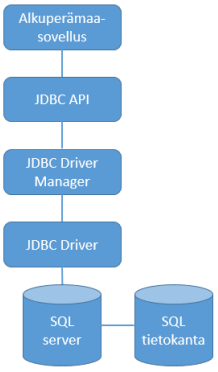
JDBC-rajapinnalla hoidetaan tietokantayhteys ajurin avulla. Ajurit toteuttavat java.sql.Driver-rajapinnan ja toimivat kerroksena servletin ja tietokannan välissä. Ajuri ottaa vastaan SQL-lauseita. Ajuri välittää lauseet tietokannalle sekä vastaanottaa tietokannasta vastauksen. Vastauksen ajuri palauttaa takaisin tietokantaa käyttävälle servletille. Ajurin kommunikoi tietokannan kanssa, riippuu ajurin tyypistä. /5/

5.16 JDBC-rajapinnan käyttäminen

Ajurin käyttämien koostuu seuravista vaiheista:

- Ladataan JDBC-ajuri
- Luodaan yhteys ajurin avulla tietokantaan
- Luodaan Statement-objekti ja suoritetaan SQL-komento
- Käsitellään komennon tuloksia
- Suljetaan yhteys.

Työssä käytetään ajurina Microsoft JDBC Driver:a. Tämä ajuri tukee Microsoft SQL Server-rajapintaa. WEB-INF/lib-hakemistoon on liitetty tarvittava sqljdbc.jar-tiedosto. (Kuvio 6.) /8/



Kuvio 6. Sovelluksen JDBC-arkkitehtuuri. /8/

5.17 Sovelluksen yhteyden luominen tietokantaan

JDBC-ajurin lataaminen täytyy suorittaa JVM:n (Java Virtual Machine, Java virtuaalikone) muistiin, jotta mahdollistetaan yhteys DriverManager-luokan avulla. Ajuri ladataan muistiin Class.forName()-kutsulla sekä se rekisteröidään. /4/

Sovelluksessa on käytetty seuraavaa ajuri-kutsua

Class.forName ("com.microsoft.sqlserver.jdbc.SQLServerDriver").newInstance();

Jos JVK ei pysty paikallistamaan ladattavaa ajuriluokkaa, aiheutta ajurin lataamien SQLException-poikkeuksen. Ajurin lataus on sijoitettu try-lohkon sisälle juuri tämän vuoksi. Ajurin latauksen jälkeen voidaan käyttää java.sql.DriverManager-luokkaa tietokantayhteyden luomiseksi. Tämän luokan getConnection()-metodin avulla luodaan yhteys tietokantaan. Tämä metodi palauttaa luokan, joka toteuttaa java.sql.Connection-rajapinnan. /4/

Sovelluksen yhteys on luotu seuraavasti:

conn=DriverManager.getConnection("jdbc:sqlserver://server;""integratedSecurity =true;");

Lauseen "integratedSecurity=true;"-osassa tarkastetaan käyttäjän tietokannan Windows-autentikointi. Tietokantoihin on annettu käyttäjälle oikeudet tiettyihin tietokantoihin ja tauluihin. /4/

5.18 SQL-komentojen suorittaminen sovelluksessa

Sovelluksessa voidaan tehdä monta kyselyä peräkkäin, joten PreparedStatement-metodi on parempi kuin Statement-metodi.

Sovelluksessa SQL-komennot on toteutettu PreparedStatement-rajapinnan toteuttavien olioiden avulla. Näiden komentojen avulla voidaan lisätä, muokata, poistaa ja hakea tietoa tietokannasta. Haku on toteutettu sovelluksessa PreparedStatement-olion avulla. PreparedStatement-olio saadaan java.sql.

Connection-oliolta preparedStatement-metodilla, jolle argumenttina annetaan SQL-lause. /4/

PreparedStatement-olio luodaan seuraavasti:

PreparedStatement ps= conn.prepareStatement(sqlQuery);

PreparedStatement -luokan executeQuery()-metodi palauttaa java.sql.ResultSet-olion. Tämä olio sisältää suoritetun haun tulokset.

Sovelluksessa kyselyn toteutus on seuraavanlainen:

```
ResultSet rs = ps.executeQuery();

	while (rs.next()) {
		ModuleCost c = new ModuleCost();
		c.setLaskuri(laskuri);
		laskuri++;
		c.setModuleName(rs.getString("ModuleName"));
		c.setComponentName(rs.getString("ComponentName"));
		c.setQty(rs.getInt("Qty"));
		c.setStockCode(rs.getString("StockCode"));
		c.setStdCostPric1(rs.getDouble("StdCostPric1"));
		c.setPurchasePric(rs.getDouble("PurchasePric"));
		c.setTotalstdcp(rs.getDouble("Totalstdcp"));
		c.setTotalPP(rs.getDouble("TotalPP"));
		c.setPros(rs.getDouble("Pros"));
		c.setUpdated(rs.getDate("Updated"));
		c.setSC01001(rs.getString("SC01001"));
		c.setSC01052(rs.getDouble("SC01052"));
		c.setSC01053(rs.getDouble("SC01053"));
		c.setSC01067(rs.getString("SC01067"));
		c.setSoo(rs.getString("soo"));
		c.setMax(rs.getDate("max"));
		c.setPurcPricMult(rs.getDouble("purcPricMult"));
```

```
                c.setSumma(rs.getDouble("summa"));
                c.setSumma1(rs.getDouble("summa1"));
                c.setSumYht(rs.getDouble("SumYht"));
                c.setMaxpros(rs.getDouble("Maxpros"));
                c.getLisays();
                c.getSum();
                c.setSC01059(rs.getString("SC01059"));
                c.setSC01058(rs.getString("SC01058"));
                c.setPL01002(rs.getString("PL01002"));
                c.setPL01001(rs.getString("PL01001"));
                      for (ModuleCost k : module) {
                            c.summa1 += (k.summa * k.lisays);
                            c.SumYht += k.summa;
                            k.pros = c.pros;
                            }

System.out.println(c.getLaskuri() + c.getModuleName() +
c.getComponentName() + c.getQty()+ c.getStdCostPric1() +
c.getPurchasePric() + c.getPurcPricMult() + c.getPurcPricMult_y()+
c.getUpdated() + df.format(c.getPros()) + c.getSC01067() + c.getSoo()+
c.getSC01058());
                module.add(c);
                            }
```

Sovelluksessa SQL-kysely haetaan kahdelta eri serveriltä ja neljästä eri taulusta. Tauluja yhdistää StockCode, joka on liitetty kaikista tauluista yhteiseksi tekijäksi. Kaikki taulut on haettu viitenimien kanssa. Viitenimi on määritelty FROM-määreessä. Taulut on liitetty toisiinsa JOIN-määreellä, jolloin StockCodet ovat liittäviä tekijöitä.

Alikyselyä käyttämällä selvitetään sovelluksessa materiaalin kustannushinnan päivitykset. Sovellus näyttää päivämäärän, jolloin materiaalin hinta on päivittynyt. Alikysely on kytketty pääkyselyyn, joten siinä käytetään ylemmän kyselyn sarakenimiä. Ryhmittely (GROUP BY) on määritelty vielä HAVING-määreellä, jonka avulla saadaan tieto uusimmasta päivästä, MAX()-koostefunktion avulla. Alikyselyssä on WHERE-määre, joka määrittää kyselyn StockCode eli minkä laitteen tiedot käyttäjä haluaa hakea.

GROUP BY-määre ryhmittelee haetut rivit siten, että kaikilla riveillä on sama arvo GROUP BY-sarakkeessa. ORDER BY-määre järjestää tulosjoukon SELECT-listan tiettyyn järjestykseen, jonka määrittelee siis ORDER BY-määreen listan järjestys.

Kyselyn tulos käydään läpi while-silmukan avulla. Tulosjoukosta haetaan aina seuraava rivi, kunnes kaikki on käyty läpi. Silmukassa on luotu Module-luokan olio Module c = new Module(). Tulosjoukosta haetaan sarakkeiden arvot ja sijoitetaan settereillä vastaaville olion muuttujille. System.out.println()-metodilla tulostetaan tulosjoukko näkyviin testausta varten. Luotu olio lisätään ArrayList-kokoelmaan add-metodilla. Suurin osa sarakkeiden nimistä tulee suoraan tietokannasta, mutta osa sarakkeista on luotu aliasnimien avulla. Aliasnimen sarakkeiden rivien arvot yhdistetään toisten sarakkeiden arvojen kanssa. Tällaisia sarakkeita ovat esim. ’summa’-, ’summa1’-, ’lisays’- ja ’SumYht’-sarakkeet.

Sarakkeen ’SumYht’ avulla lasketaan materiaalin valmistuskustannukset. Sarakkeen rivien yhteenlaskettu arvo saadaan for-silmukassa lisäämällä rivi riviltä yhteenlaskettu arvo edelliseen arvoon. Yksikkökustannushinta jaetaan pakkauskappalemäärällä sekä tulos kerrotaan laitteen sisältämän kyseisen materiaalin kappalemäärällä. Tulokseksi saadaan juuri tämän materiaalin yksikkökustannushinta. Sarakkeiden rivien arvojen yhteenlasketulla arvolla saadaan koko taajuusmuuttajan kaikkien materiaalien kustannushinta.

Sarakkeen ’lisays’ avulla muutetaan alkuperämaamerkkijono (’soo’) binääriseksi luvuksi, 1 tai 0. Tätä saraketta käytetään hyväksi laskettaessa NonEU-osuus sarakkeessa ’summa1’, summan ja lisayksen tulolla. EU-osuus saadaan sarakkeiden ’SumYht’ ja ’summa1’ erotuksesta. Sarake ’summa’ on materiaalien yksikköhinnat ilman koko sarakkeen rivien yhteenlaskettua arvoa.

Sovelluksen yhteyden sulkeminen kyselyn jälkeen on tärkeää, koska tällöin ei yhteys ns. jää päälle ja kuluta resursseja turhaan.

Metodeilla initDB(), openDB() ja closeDB() on huolehdittu tietokantayhteyden toimivuudesta. initDB()-metodilla avataan tietokantayhteys ja luodaan PreparedStatement-luokan stmt-olio. openDB()-metodin avulla ladataan ajuri ja alustetaan tietokantayhteys. closeDB()-metodilla suljetaan tietokantayhteys. Kaikissa kolmessa metodissa on käytetty try-catch-rakennetta virhetilanteiden vuoksi. /4/

Metodin perässä on ”throws SQLException”-osa, jota sanotaan poikkeuksen heittämiseksi kutsuketjussa ylöspäin. Poikkeuksen sattuessa siepataan virhetilanne ja try-catch-rakenne hoitaa ongelman ohjelmoijan haluamalla tavalla. Try-osio sisältää tapahtuman, josta voi muodostua poikkeus eli määritellään mahdollinen poikkeuksen aiheuttaja. Jos poikkeusta ei tapahdu, käydään try-osio läpi, mutta jätetään catch-osio läpikäymättä. Catch-osiossa määritellään poikkeuksen tyyppi ja poikkeuksen nimi sekä poikkeus käsitellään täällä, eli try-osio aiheuttaa poikkeuksen ja catch-osio huolehtii sen käsittelystä. /4/

Try-catch-rakenteessa voi olla monta catch-osoita erityyppisten virhetilanteiden sieppaamiseen, eri Exception-luokan oloiota. Lauseessa voi olla myös finally-osa, joka suoritetaan, riippumatta siitä onko catch-osiota suoritettu vai ei. /4/

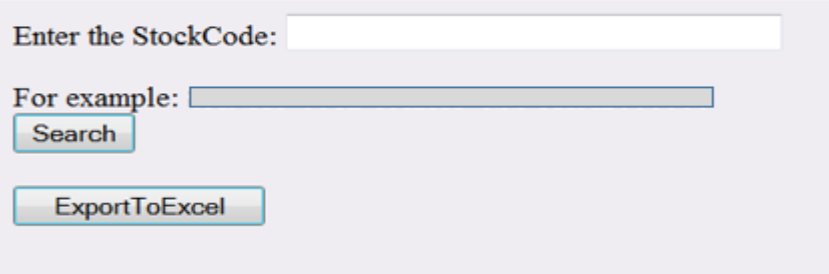
```
finally {
	if (ps != null) {
		try {
			ps.close();
		} catch (SQLException e) {

		}
	}
	if (conn != null) {
		try {
			conn.close();
		} catch (SQLException e) {

		}
	}
```

Yhteyden sulkeminen varmistetaan finally-osiossa. Tietokantayhteyden ja PreparedStatementin sulkemien on hyvä varmistaa, että ne eivät syö resursseja muilta ja eivät ole turvallisuusriski jäädessä auki. Kun nämä tehdään finally-osiossa, varmistetaan niiden sulkeminen myös mahdollisessa virhetapauksessa. /4/

5.19 Sovelluksen käyttöliittymä

Käyttöliitymä on rajapinta, jolla sovellus esitetään käyttäjälle. Käyttöliittymältä vaaditaan helppokäyttöisyyttä, helposti omaksuttavuutta, logiikaltaan samanlainen joka osioissa sekä toimivuutta, joka on hyvä ja luotettava. Käyttöliittymän tulee palvella kohderyhmäänsä hyvin ja monipuolisesti.

Sovelluksen käyttöliittymän alkusivu sisältää syöttölaatikon StockCoden syöttämistä varten. Search-painike hakee tiedot näytölle. Toinen mahdollinen valinta on kyseisen tulosjoukon Excel-tiedostoon vienti (Kuvio 7.).

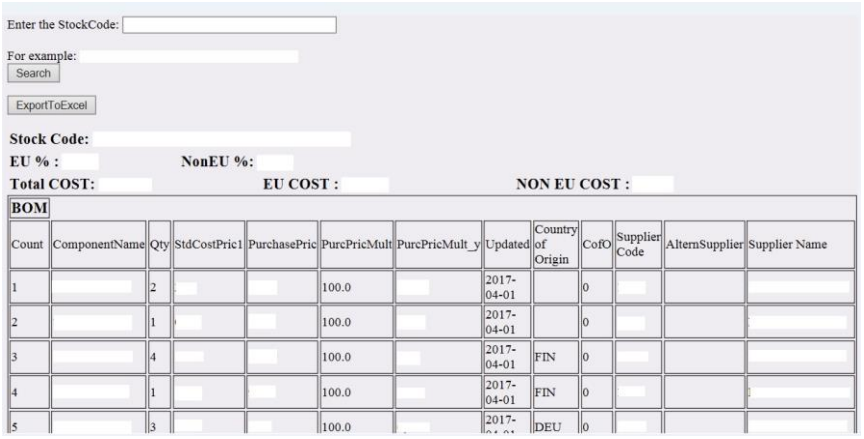


Kuvio 7. Käyttöliittymän alkusivu sovelluksessa.

Sovelluksessa käyttöliittymä on toteutettu servletissä. Servletti hakee tiedot Dao-luokasta ja tulostaa näkymän HTML-sivuille. Tiedot tuodaan taulukkomuotoon näytölle ja lisätiedot tuodaan sivun alkuun yksittäisinä tietoina.

Ensimmäisenä tulostuu näkyviin itse StockCode, joka on koko haun ydin. Yksittäiset tiedot ovat materiaalikustannusten EU-prosenttiosuus sekä EU:n ulkopuolinen prosenttiosuus, NonEU-osuus. Materiaalin kokonaiskustannus, materiaalikustannus EU ja materiaalikustannus NonEU näkyvät myös yksittäisissä tiedoissa euromääräisenä.

Tuotteen BOM (Bill of Material) eli tuotteen osaluettelo on taulukkomuodossa yksittäisten tietojen alla. Sarakkeina taulukossa näkyy laskuri, komponentin nimi, määrä laitteessa, yksikköhinta materiaalille, myyntihinta materiaalille, pakkauskoko (kpl), laskettu yksikköhinta, päivityspäivä, valmistusmaa, valmistusmaa binäärisenä, alihankkijan numerokoodi, alihankkijan nimi.


Enter the StockCode:

For example:

Search

ExportToExcel

Stock Code:

EU % : **NonEU %:**

Total COST: **EU COST :** **NON EU COST :**

BOM

Count	ComponentName	Qty	StdCostPric1	PurchasePric	PurcPricMult	PurcPricMult_y	Updated	Country of Origin	CofO	Supplier Code	AlternSupplier	Supplier Name
1		2			100.0		2017-04-01		0			
2		1			100.0		2017-04-01		0			
3		4			100.0		2017-04-01	FIN	0			
4		1			100.0		2017-04-01	FIN	0			
5		3			100.0		2017-	DEU	0			

Kuvio 8. Tulosjoukon näkymä asiakkaalle.

NonEU-osuus tuotteen materiaalista saadaan summalla, josta on poistettu EU-alkuperämaan osuus. Koska NonEu:n binääriluku on 1 ja EU:n binääriluku on 0, saadaan NonEU-osuus kokonaiskustannushinta kerrottuna binääriluvulla 0. Tulokseksi saadaan NonEU-osuus. EU-osuus on kokonaiskustannushinnan ja NonEU-osuuden erotuksesta.

Laitteen kokonaiskustannus euromääräisenä saadaan suoraan sarakkeen rivien yhteenlaskusta kerrottuna materiaalin kappalemäärällä. NonEU-osuus saadaan edellä laskettu ’summa1’-sarakkeesta kerrotuna materiaalin kappalemäärällä. Tästä saatu tulos vähennetään kokonaiskustannuksista, josta saadaan siten EU-osuus euromääräisenä.

Laskuri laskee vain rivien määrän komponenteille. Komponentit ja niiden määrä määräytyvät suoraan laiteen StockCodesta. Yksikkökustannushinta määräytyy yksittäiselle materiaalille. Myyntihintakustannus jaetaan kokonaispakkauksessa olevien materiaalien määrällä ja tämä tulos kerrotaan vielä kappalemäärällä. Kappalemäärä on se, kuinka monta kappaletta laitteessa on kyseistä materiaalia. Päivityspäivä on se päivä, jolloin materiaalin päivitys on tehty. Alkuperämaa näytetään ensin merkkijonona ja seuraavassa sarkkeessa (CofO) sama asia näytetään binäärilukuna eli 0 tai 1. Kolme seuraavista sarakkeista liittyy alihankkijan tietohin: alihankkijan numeerinen koodi, numeerinen koodi vaihtoehtoiselle alihankkijalle samasta materiaalista ja alihankkijan nimi (Kuvio 8.).

6 SOVELLUKSEN TESTAUS JA KÄYTTÖÖNOTTO

Testauksen tukena on käytetty nykyistä dataa, joka on Excel-tiedostossa. Sovelluksen tuloksia on verrattu siihen. Tulokset on myös laskettu manuaalisesti verraten tuloksia keskenään. Prosenttituloksessa eroa ei ole. Euromääräisessä laskelmassa ero on noin 10 % verran. Ero selittyy sovelluksen desimaalilukujen tarkkuudella. Sovelluksen desimaalitarkkuus on tuhannesosien tarkkuudella ja nykyinen laskenta Excel-tiedostossa on sadasosien tarkkuudella. Sovelluksen desimaalitarkkuus on tarkempi kuin nykyinen tarkkuus Excel-tiedostossa. Sovelluksen desimaaliluvussa on neljä merkitsevää numeroa ja nykyisessä laskentatavassa on kaksi merkitsevää lukua.

Esimerkkinä erään materiaalin kappalehinnan ero. Kappaleen hinta on esitetty Excel-tiedostossa hintahaussa 0,0265 euroa, juuri kuten sovelluksessakin, mutta ero saadaan Excel-tiedostossa laskiessa kappaleen hintaa laitteen kokonaisuuden yhteydessä. Hinta pyöristyy silloin 0,03:een.

$$prosenttikerroin = 0{,}0300 - 0{,}0265 = 0{,}0035 \quad (1)$$

$$prosentti\ p = \frac{0{,}0035}{0{,}0300} * 100\% = 11{,}67\% \quad (2)$$

Laskutulos osoittaa, että yksittäisen materiaalin hinta sovelluksessa on yli 10 % pienempi kuin Excel-tiedoston pyöristetty desimaaliluku. Tarkuus on yhden materiaalin osalta pieni, mutta kokonaistuloksessa näkyy jo euromääräisenä tuloksena. Prosenttiosuus on sama molemmissa tuloksissa.

Testaukessa käytettiin monta eri modulinimikettä tarkastaessa Excel-tiedostosta tulosten paikkansa pitävyydet. Testauksessa sovellus antoi asiakkaan vaatimuksen mukaiset tulokset.

Käyttöönotto tapahtui heti, kun sovelluksen testaus oli varmistettu oikeaksi. Käyttäjät hyväksyivät sovelluksen toimivuuden.

7 YHTEENVETO

Asiakkaat ovat käyttäneet ennen sovellusta Excel-tiedostoa, joka on päivitetty tietokannasta. Päivityksen mukana on tullut paljon ylimääräistä raakadataa, joka on manuaalisesti piilotettu Excel-tiedostossa. Data on käsitelty siis manuaalisesti halutulla tavalla.

Sovelluksen tarkoitus on tuoda lisäarvoa käyttäjille. Sovelluksen lisäarvo tuottaa kuukaudessa kolmen tunnin etuuden sen varsinaiselle omistajakäyttäjälle. Manuaalinen työ jää pois ja tieto saadaan heti käyttöön sekä se voidaan tallentaa Excel-tiedostoon myöhempää tarkastelua varten.

Java Servlet-tekniikka tukee hyvin sovelluksen käyttötarkoitusta tiedon saatavuuteen HTML-sivuille nopeasti ja ketterästi ilman mitään ylimääräistä raakadataa. Kolmikerrosarkkitehtuuri keventää asiakaskerroksen taakkaa sekä on nopea ja ketterä tiedonhakuprotokolla.

Sovelluksessa olisi voinut käyttää JPA:ta (Java Persistence Api) , mutta tietokantataulut eivät tukeneet niitä kovin hyvin. Taulut sisältävät paljon sarakkeita, jotka pitää minimoida tiedon nopeaa saantia varten. Sovelluksessa ei lisätä eikä poisteta mitään dataa tauluista.

Sovelluksesta on pyydetty päivityksiä, jotka tekevät sovelluksesta vielä monimuotoisemmaksi tulevaisuudessa.

LÄHTEET

/1/ Danfoss Drives. Viitattu 13.3.2017. http://www.danfoss.fi/home/#/

/2/ Tavaran yleinen alkuperä. 2017. JavaTCenter. Viitattu 27.3.2017.

http://tulli.fi/yritysasiakkaat/tuonti/tavaran-yleinen-alkupera1

/3/ Hovi, A. 2012. SQL-opas. Jyväskylä. Docendo Oy.

/4/ Ahonen, T., & Hämeen-Anttila, T. & Åstrand, K. 2003. Java Servlets. Jyväskylä. Docendo Oy.

/5/ Kuha, J. 2008. Tehokas Java EE-sovellustuotanto. Jyväskylä. Docendo Oy.

/6/ Haikala, I., & Märijärvi, J. 2006. Ohjelmistotuotanto. Helsinki. Talentum Media Oy.

/7/ Hunter, J., & Crawford, W. 1998. JAVA Servlet Programming. Sebastopol. O'Reilly.

/8/ Using the JDBC Driver. 2017. Microsoft JDBC Driver. Viitattu 27.3.2017.

https://docs.microsoft.com/en-us/sql/connect/jdbc/using-the-jdbc-driver